CLAIMS

We claim:

1. A method of job dispatch in a queuing system, comprising:

   Aliasing each worker in a plurality of workers to a same or different node in a tree hierarchy;

   When a job is submitted for processing, filtering the workers to produce an unsorted, eligible worker list;

   Searching the eligible worker list to match a job with a worker by comparing a node weight, and optionally priority, of the job with the nodes;

   Selecting a node for the job based optimal comparison;

   Reserving a worker aliased to the selected node; and

   Dispatching the job to the reserved worker.

2. The method of claim 1, wherein if the reserved worker fails to process the job, the worker is removed from the eligible worker list, and another iteration of the method of claim 1 proceeds, starting from the third step of claim 1.
3. The method of claim 2, wherein each worker reserved in successive iterations of claim 2 fails to process the job, and the number of failures cumulates to a system-defined limit, then the eligible worker list is sorted, and the job is dispatched to the first ranked worker on the sorted worker list.
4. The method of claim 1, wherein a check environment process associated with the job executes on the reserved worker after the step of dispatching the job to the reserved worker, which process compares job attributes with worker attributes, and if the job attributes do not acceptably match the worker attributes, the worker rejects the job, the worker is removed from the eligible worker list, and another iteration of the method of claim 1 proceeds from the third step of claim 1.
5. The method of claim 4, wherein each worker reserved in successive iterations of claim 3 rejects the job, and the number of rejections cumulates to a system-defined limit, then the eligible worker list is sorted, and the job is dispatched to the first ranked worker on the sorted worker list.
6. A method of job dispatch in a queuing system, comprising:

   Aliasing each worker in a plurality of workers to a same or different node in a tree hierarchy;

When a worker becomes available for processing a job, filtering jobs to produce an unsorted, eligible job list;

Searching the eligible job list to match a job with a worker by comparing a node weight, and optionally priority, of the job with the node to which the worker is aliased;

Selecting a job for the worker's node based optimal comparison;

Reserving the worker; and

Dispatching the job to the reserved worker.

7. The method of claim 6, wherein if the reserved worker fails to process the job, the job is removed from the eligible job list, and another iteration of the method of claim 6 proceeds, starting from the third step of claim 6.
8. The method of claim 7, wherein each job dispatched to the worker in successive iterations of claim 7 results in the worker's failure to process the job, and the number of failures cumulates to a system-defined limit, then the eligible worker job is sorted, and the first ranked job on the sorted job list is dispatched to the worker.
9. The method of claim 6, wherein a check environment process associated with the job executes on the reserved worker after the step of dispatching the job to the reserved worker, which process compares job attributes with worker attributes, and if the job attributes do not acceptably match the worker attributes, the worker rejects the job, the job is removed from the eligible job list, and another iteration of the method of claim 6 proceeds from the third step of claim 6.
10. The method of claim 9, wherein each job dispatched in successive iterations of claim 9 results in the worker's rejection of the job, and the number of rejections cumulates to a system-defined limit, then the job list is sorted, and the first ranked job on the sorted job list is dispatched to the worker.
11. A queuing system comprising at least one client, at least one supervisor, and at least one worker, together with network communications between each client and supervisor and between each supervisor and worker, and a component selected from the group comprising EDQS messaging architecture, EDQS event-driven dispatch, EDQS event/callback architecture, EDQS job type data architecture, and the EDQS domain specific language.
12. The queuing system of claim 11, wherein the job type data architecture comprises a job type element containing a descriptor datum, an executor datum, a GUI name datum, and icon datum, a commander datum, binding scripts, names of associated libraries, and name of index file.

13. The job type data architecture of claim 12, further comprising a job type directory that contains the job type element, together with code files identified in the job type element.
14. The queuing system of claim 11, wherein the messaging architecture uses messages written in the EDQS domain specific language.
15. The queuing system of claim 11, wherein the domain specific language uses a message generator that converts platform-specific commands into a data structure that can be interpreted by other types of platforms.
16. The queuing system of claim 11, wherein the event/callback architecture is based on enabling triggers whenever predefined Boolean expressions of event states are true, which enabled triggers cause callback code to be executed by a supervisor.
17. The event/callback architecture of claim 16, wherein the callback code causes a job-driven dispatch.
18. The event/callback architecture of claim 16, wherein the callback code causes a worker-driven dispatch.
19. The event/callback architecture of claim 16, wherein the trigger evaluates Boolean expressions of event states of one or more jobs in a process group.
20. The event/callback architecture of claim 16, wherein the event/callback architecture uses evname, evtype, and evcontext variables to define an event.